Yasca User's Guide

Table of Contents

1 Introduction to Yasca

1.1 Purpose

Yasca was created to help software developers ensure that applications are designed and developed to meet the highest quality standards. It is related to QA testing and vulnerability scanning, but replaces neither. Instead, Yasca can be used during development to catch much of the "low hanging fruit" that may only be found much later in the development lifecycle. Distributed with both custom scanners and embedded open-source tools (e.g. JLint, antiC, Lint4J, FindBugs, and PMD, Yasca is able to deliver a relatively comprehensive analysis of scanned applications.

Yasca can be thought of as an aggregation tool "plus a little more". While the majority of detected issues are actually found by the open-source tools, the "little more" consists of plugins written for Yacsa to detect issues that the other tools did not scan for.

The philosophy behind Yasca is that developers should have access to a suite of tools to enable them to better develop secure software. Since much work has been done in this area in the form of disparate products performing similar functions, it was important to aggregate the results back in a simple, easy-to-use tool.

1.2 Scope

This guide is meant to be both a user manual and a developer guide for extending Yasca. It does not contain detailed information on any of the embedded tools that accompany Yasca. (This information resides in the **docs** directory.)

1.3 System Organization

Yasca is distributed as a lightweight, "no-install" desktop application. Most components are written in an included minimal distribution of PHP v5.2.5. As with all products embedded in Yasca, future distributions will attempt to include the latest stable release.

Logically, Yasca consists of a basic front-end, a set of scanning plugins and report renderers, and an engine to tie them all together.

Yasca has two front-ends available: a command line interface and a Windows GUI. Each can be used to start Yasca, but the command line interface has considerably more options available and offers better performance.

All plugins reside in the **plugins** directory. Three plugins are included in Yasca that have external dependencies. The first is **JLint**, which scans Java .class files, and requires the **jlint.exe** file to be available in the **resource/utility** directory. The second is antiC, which scans Java and C/C++ source code, and also requires the **antic.exe** file to be available in the **resource/utility** directory. The third is **PMD**, which partially compiles Java source code and scans the resulting abstract syntax tree. PMD requires Java JRE 1.4 or later. If any of these dependencies cannot be located, Yasca will issue a warning but will continue scanning with other plugins.

The output of Yasca is a file created by a specific report renderer. Yasca currently has renderers for rich HTML, XML, and CSV formats.

The Yasca engine, which ties the other components together, is embedded in distributed binaries. It is not meant to be modified except as part of a subsequent release. (This differs from plugins, which are meant to be modified as needed.)

2 Installation & Use

2.1 First-time Users

Before using Yasca, be sure your system meets the following prerequisites:

- Microsoft Windows 2000 or later**
- Java JRE (or JDK) v1.4 or later (required for PMD)
- At least 512 MB RAM (1+ GB suggested)

Note: Yasca was not tested on a wide range of environments, so if you run into trouble, please let us know.

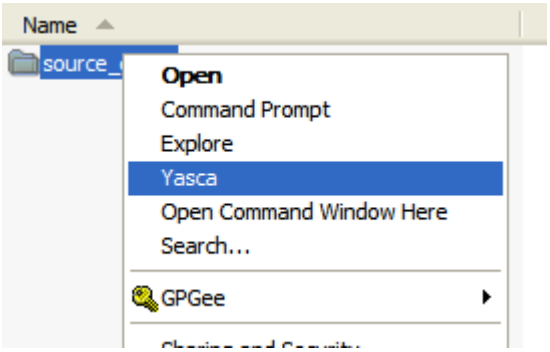
Two methods of using Yasca are available: **local installation** and **network installation**.

2.1.1 Local Installation

Step 1: Download the Yasca binary from http://yasca.org/ and install it to a directory of your choice.
Step 2: Test the installation by running the following command:
(Unix) **yasca ./resources/test/**
(Windows) **yasca resources\test**
Step 3: Check the new file created on your desktop in the **Yasca** folder (Yasca-Report-NNNNNNNN-NNNN.html). It should contain findings from the test.

2.1.2 Shell Installation

You can integrate Yasca into the Windows Explorer shell by running the **etc/yasca.reg** registry file.



The yasca.reg file is set up to look for Yasca at the following location:

c:\dev\yasca

If you want to change that location, you can edit the registry manually. The setting is located at:

HKLM\SOFTWARE\Classes\Folder\shell\Yasca!\command

2.1.3 Non-Windows Installations

Although Yasca was developed and tested under Windows, only a few components are Windows-specific, such as the JLint and antiC plugins, and the batch files that initiate the Java-based tools. It is expected that a future version will remove this dependency and make it easy to run on non-Windows systems.

2.2 Licensing

Yasca is being made freely available to members of the community under a BSD license.

2.3 Installing the system

Describe the procedures that the user must perform so they can access/install software, configure software, delete data, and setup software operations.

2.4 Starting the system

2.4.1 Starting from the Command Line

Yasca can be started from the command-line by typing yasca.exe from the installation directory. Appropriate command line usage should be displayed.

In general, most users should simply run yasca <directory>. This will output an HTML report to the user's desktop. If this report location is inappropriate, the --output argument can be used to direct output to a different location.

2.4.1.1 Command Line Options

2.4.1.2 Alternative Scenarios

This section will describe various other scenarios that may be helpful in utilizing Yasca to it's fullest potential.

Only Run One Plugin:

```
yasca -p plugins/PluginName.php <target directory>
```

Run All But One (or more) Plugin(s)

```
yasca -px FindBugs,PMD <target directory>
```

2.5 Stopping Yasca

You can stop Yasca from the command line interface at any time by pressing CTRL-C. If an external process (such as **Java**) has been forked, you may need to kill it separately (via the task manager or the **`kill`** command).

3 Plugins

3.1 About Plugins

Yasca uses individual plugins to perform the actual scanning of targeted files. This design allows Yasca to be easily extended to scan additional file types as needed. "Plugin Packs" can be created and distributed to scan specific file types, or issues like security, performance, or complexity.

Yasca is distributed with both external plugins as well as a number of native plugins. Though there is no difference between the two from the engine's perspective, we define a native plugin to be one that is completely self-contained and extends the Plugin class, and we define an external plugin to be everything that requires additional software for processing.

The external plugins distributed with Yasca are:

- **Grep Plugin.** Uses external GREP files to scan target files for simple patterns.
- **PMD Plugin.** Uses PMD to parse and scan Java (and JSP) source code for issues.
- **JLint Plugin.** Uses J-Lint to scan Java .class files for issues.
- **antiC Plugin.** Uses antiC to scan Java and C/C++ source code for issues.
- **FindBugs Plugin.** Uses FIndBugs to scan Java class and Jar files for issues.
- **Lint4J Plugin.** Uses Lint4J to scan Java .class files for issues.

All plugins are located within the **plugins** directory, but this can be overridden at runtime. See the section describing command-line arguments for more information. The plugin can reside directly in the plugins directory or within any directory beneath it. Tools used by the standard plugins are located in the **resources/utility** directory.

Each of the plugins is designed with the concept of scannable file types (extensions). A plugin will only be invoked on files that it is capable of scanning. For instance, a plugin that scans Java source code can be configured to only scan files with the .java extension.

Certain file extensions are ignored for all plugins by default. These extensions include **exe**, **zip**, **jpg**, **gif**, **png**, and **pdf**. These can be overridden by using the `--ignore-ext` command line option. To exclude no files (i.e. include all files), use `--ignore-ext 0`.

Note: You can disable a plugin, GREP file, or PMD ruleset by prefixing the filename with an underscore. For example, rename 'MyPlugin.php' to '_MyPlugin.php', or by using the -px command line option.

Each of the plugins assign a severity rating to each discovered issue. In order to keep the upgrading process as simple as possible, those ratings should not be changed by the user. Instead, the file **reources/adjustments.xml** can be used to change the rating or description from any plugin. For example,

if you wanted to change the severity level of findings generated by the PMD plugin (under the rule "EmptyCatchBlock") , you could add an entry to the file like this:

```
<adjustment plugin_name="PMD" finding_name="EmptyCatchBlock" severity="5"/>
```

Alternatively, if you wanted to increase the severity by one (i.e. making it less severe), you could have made it:

```
<adjustment plugin_name="PMD" finding_name="EmptyCatchBlock" severity="+1"/>
```

You can also change the description by including a <description> element, as in:

```
<adjustment plugin_name="PMD" finding_name="EmptyCatchBlock">
      <description>Some New Description</description>
</adjustment>
```

You can append or prepend a description by using the method attribute:

```
<adjustment plugin_name="PMD" finding_name="EmptyCatchBlock">
    <description method="append">Some Extra Information Here</description>
</adjustment>
```

3.2 Plugin Details

This section describes each of the plugins included in the Yasca distribution. This does not include PMD rulesets or the issues that JLint and antiC find. These are located in the external documentation.

3.2.1 Grep Plugin

The Grep Plugin uses external files (*.grep) located in the plugins directory to scan target files for patterns. The format of a GREP file is as follows:

```
name = <Name of the Plugin>
file_types = <comma,separated,extension,list>
pre_grep = /<regular expression>/    (optional)
grep = /<regular expression>/
lookahead_value = 10
category = <Category Name>
severity = <Severity (1-5)> (optional)
category_link = <URL for information about category> (optional)
description = <Description of the finding> (optional)
```

As an example, below is the **Process.ForName.grep** file:

```
name = Use of Class.forName()
```

```
file_type = java,jsp
grep = /Class\.forName\(/
category = Process Control
severity = 3
category_link = http://www.fortifysoftware.com/vulncat/java/...
```

As you can see, the grep statement above requires that a valid PCRE-style regular expression be used, enclosed within '/' characters. You can also use modifiers such as 'i' after the closing '/'.

The Grep Plugin is distributed with the following GREP files:

Grep File	Category	Description
Ajax.grep	Non-standard Technology	Detects AJAX use in JSPs.
Authentication. SimplePassword.grep	Authentication	Using a simple password.
Authentication. StoredPassword.grep	Authentication	Storing a cleartext password in an object
Authorization.Debug.grep	Authorization	Using a 'debug' parameter
Bug.JavaScript. InternalAndExternal.grep	Bug	Using both a SRC= as well as inline JavaScript in the same tag.
Bug.JavaScript. ScriptTag.grep	Bug	Using a <script> tag in a .js file.
Console.Output.grep	Poor Logging Practice	Use of System.[out|err].print(ln)
Crypto.XOR.grep	Weak Cryptography	XOR-encryption
Crypto.MD5.grep	Weak Cryptography	Weak hash function
Crypto.Random.grep	Weak Cryptography	Weak source of randomness
CustomCookies.grep	Information Disclosure	Only SessionID should be sent.
DoS.ReadLine.grep	Denial of Service	ReadLine blocks until EOF is found
Error-Handling.StackTrace. JSP.grep	Error Handling	Printing a stack trace in JSP
Formatting.MissingAMPM.grep	Bug	Missing AM/PM when printing in 12-hour time.
General.BadLanguage.grep	Code Quality	Looks for bad words.
General.Password. Hardcoded.grep	Weak Authentication	Username == Password
General.NonProduction	Miscellaneous	Using "Hello World" code.
Information-Disclosure. Comment.grep	Information Disclosure	Comments in HTML
Injection.FileInclusion.grep	Injection	Including arbitrary files
Injection.SQL.grep	Injection	Possible SQL Injection
Injection.XSS.JSP.grep	Injection	Cross Site Scripting in JSPs
Licensing.grep	Licensing	Inclusion of GPL, etc. code
Process.exec.grep	Process Control	Dangerous function call
Process.ForName.grep	Process Control	Dangerous function call
Process.LoadLibrary.grep	Process Control	Dangerous function call
Stability.Sleep.grep	Stability	Servlets are singletons, sleep = block

String.equals.grep	Code Correctness	foo == "bar" => incorrect usage

3.2.1.1 The pre_grep Command

The pre_grep command allows you to specify a regular expression that must match a line within N lines of a regular expression matched within the grep expression. For instance consider:

```
pre_grep = /foo/
grep = bar
```

This line would look for lines matching /bar/, but only if another line matching /foo/ were found within 10 lines before it. The "10 lines" in this case refers to the default value of **lookahead_value**, which can be also be specified in the .grep file.

One difference between pre_grep and grep: Only one **pre_grep** expression can be used, while multiple **grep** expressions can be.

3.2.2 PMD Plugin

The PMD Plugin uses the open-source tool PMD to partially compile Java source code and JSP files and then scan the resulting abstract syntax tree for certain patterns. It is extremely powerful, but can only operate on Java (and JSP) source code.

Warning: PMD is particularly fragile when scanning JSP files. Since different

PMD uses rules defined externally through a Java class file or an XPath expression. These rules (or references to rules) are grouped into rulesets which are provided in the **plugins/default/pmd** directory. Currently, the only active plugin is **yasca.xml**, but that plugin references other rulesets. It is recommended to continue using this model wherein only a small number of active PMD plugins are invoked, and that they in turn reference the specific rules and rulesets to be included.

Documentation on PMD is available at the PMD website and from the book "Applied PMD".

3.2.3 JLint Plugin

The JLint Plugin uses the open source tool JLint v3.0 (http://jlint.sourceforge.net/) to scan compiled .class files, searching for bugs, inconsistencies, and synchronization problems.

The JLint user manual provides a listing of everything JLint searches for, and is distributed in the **docs/3rdparty** directory.

3.2.4 antiC Plugin

The antiC Plugin uses the open source tool antiC v1.11.1 (http://jlint.sourceforge.net/) to scan Java and C/C++ source code, searching for bugs, inconsistencies, and synchronization problems.

The JLint user manual provides a listing of everything antiC searches for, and is distributed in the **docs/3rdparty** directory.

3.2.5 Minor Plugins

Minor plugins have been included when a pattern was too complex to match using a simple regular expression. Each of these plugins are described below.

3.2.5.1 Weak Authentication (Authentication.Weak.php)

Weak authentication is detected when nearby lines look like they define a username and a password to be the same value.

```
// Startup Properties

database.conn.username = tango44opine
database.conn.password = tango44opine
```

3.2.5.2 Redundant Null Check (Code-Quality.Null.Redundant.php)

A redundant null check occurs when a particular expression is checked for null directly after it is already used in a context where an exception would have been thrown if it were. This may be legitimate: the function bar() below could modify a global object that contains foo, setting it to null; or a NullPointerException could be caught and handled appropriately.

```

foo.bar();
if (foo != null) {
...
}
```

3.2.5.3 Empty Catch Block (Error-Handling.Catch.Empty.php)

In many circumstances, ignoring thrown exceptions can be the source of application instability. This plugin checks to expressions like the one below:

```
try {
...
```

```
} catch(Exception ex) { }
```

This plugin is duplicated by the PMD rule **basic/EmptyCatchBlock** and may be removed in a future version.

3.2.5.4 Non-Standard/Outdated Libraries (File-System.Non-Current-Libraries.php)

This plugin scans all library files (.jar, .so, .dll) to see if they match an internal whitelist of "current" libraries, stored in **resources/current_libraries/**. The files in this directory contain a list of "acceptable" libraries. Anything not in that list will be flagged.

3.2.5.5 Temporary Files (File-System.Temporary-Files.php)

This plugin checks the filename for what appears to be a temporary file naming convention (e.g. 'tmp', 'temp', 'dummy', or a prefix of '~$').

3.2.5.6 Cross Site Scripting: Simple via Servlets

This plugin attempts to find simple examples of cross-site scripting within a servlet.

```
class FooServlet extends Servlet {
public void doPost(HttpServletRequest req, HttpServletResponse res) {
  String bar = req.getParameter("bar");
  ...
  out.println("bar = " + bar);
}
}
```

3.2.5.7 External E-Mail Address (Information-Disclosure.Email.External.php)

This plugin attempts to find external e-mail addresses embedded in source code or other files. This is not strictly a problem, but licensing considerations should dictate whether it is appropriate to include third party libraries and functions within applications.

The list of "internal" domain names are defined in the plugin file.

More information is available at:

- http://creativecommons.org/
- http://www.gnu.org/copyleft/gpl.html
- http://en.wikipedia.org/wiki/Open-source_license

3.2.5.8 COBOL Resource Leakage (Code-Quality.Resource-Leak.GETMAIN.php)

This plugin scans for potential resource leaks in COBOL source code. The resource leak in question comes from executing a GETMAIN without an associated FREEMAIN. Long running jobs can exhaust available resources.

```
EXEC CICS GETMAIN                                          00010000
          SET (ADDRESS OF SOME-VARIABLE)                   00010010
        LENGTH (LENGTH OF SOME-VARIABLE)                   00010020
        NOHANDLE                                           00010030
END-EXEC                                                   00010040
```

3.3 Writing Your Own Plugins

It is easy to extend Yasca by writing your own plugin. This can take the form of a PHP script, GREP file, or PMD ruleset placed in the **plugins** directory.

3.3.1 Writing a New PHP Plugin

Writing a new PHP Plugin means extending the Plugin class (see the API or Plugin.php source code), implementing an execute() function that will perform the scan when invoked.

Suppose you want to write a PHP Plugin to search for all social security numbers hardcoded in Java source code or .properties files. The first thing you need to do is define what type of plugin it is.

There are two types of plugins: **single-target** and **multi-target**. Single targets operate only on the file passed from the engine, and multi-targets generally operate on all files in the target directory at once. The Yasca engine is not aware of this distinction, so multi-target plugins will be invoked once for each target file. It is therefore important to prevent the plugin from executing multiple times, which can be done like this:

```
function execute() {
    static $alreadyExecuted;
    if ($alreadyExecuted == 1) return;
    $alreadyExecuted = 1;
```

Next, you should decide on the specific file types that will be scanned. In this case, the file extensions .java and .properties should suffice.

The last step is writing the code. The example below shows the finished product. Embedded comments have been removed for brevity, but full documentation should be included in any plugins that you create.

```
001 <?php
```

```
002 class plugin_ssn_search extends Plugin {
003  var $valid_file_types = array("java", "properties");
004
005   function execute() {
006     $pat = "[^\d]\d{3}-?\d{2}-?\d{4}[^\d]";
007     if (preg_grep('/' . $pat . '/', $this->file_contents), $matches) {
008       foreach ($matches as $line_number => $match) {
009          $result = new Result();
010          $result->line_number = $line_number;
011          $result->severity = 5;
012          $result->category = "Social Security Number";
013          array_push($this->result_list, $result);
014       }
015     }
016   }
017 }
018 ?>
```

We will now drill down into this examine and explain all important parts of this file.

```
002 class plugin_ssn_search extends Plugin {
```

Line 002 defined the class name to be `plugin_ssn_search`, which extends `Plugin`.The class name must be derived from the name of the file that it is defined in. The Yasca engine scans the **plugins** directory for all PHP files, makes a note of the file name, and executes an include() on the file. The file name noted is transformed to match the expected class name by replacing all hyphen (-) and period (.) characters with an underscore (_), and converting the entire string to lowercase. Therefore, the plugin file **Foo-Bar.Quux.php** would be expected to have the class **plugin_foo_bar_quux**.

```
003  var $valid_file_types = array("java", "properties");
```

This line defines the valid file types that can be scanned. In this case, we are only scanning Java source code and .properties files. In order to scan all files, simply use an empty array.

```
005   function execute() {
006     $pat = "[^\d]\d{3}-?\d{2}-?\d{4}[^\d]";
007     if (preg_grep('/' . $pat . '/', $this->file_contents), $matches) {
008       foreach ($matches as $line_number => $match) {
```

Lines 005 through 016 show the implementation of the `execute()` function. This function overrides the `Plugin::execute()` function (which does nothing itself). A regular expression is defined in line 006 and is matched on line 007 against `$this->file_contents`, which is an array pre-populated with the full text contents of the file being scanned. On line 008, we loop over each match found from the array.

```
009          $result = new Result();
010          $result->line_number = $line_number;
011          $result->severity = 5;
012          $result->category = "Social Security Number";
013          array_push($this->result_list, $result);
```

On lines 009 through 013 we create and use a `Result` object to pass the results of the scan back to the engine. The Result object has a number of properties (see the API documentation for details), but each will be filled in with defaults. The use of the specific properties is important, however, so they will be explained.

Variable	Description
`$result->line_number`	Used by HTMLReport to include a snippet of from the file in the final report. Also included in other reports. Warning: The `$this->file_contents` object is indexed from 0, whereas line numbers start at 1. You should generally add 1 to any index used. This defaults to 0. **Note:** If you set the `line_number` to 0, then a snippet will not be included in the final report.
`$result->severity`	The severity is an integer value in the range of 1-5, meaning the following: 1. Critical 2. High 3. Medium (default) 4. Low 5. Informational The severity affects the initial sorting of the reports as well as filtering that can be done with the --level command line option.
`$result->category`	This should contain a brief description (only a few words) of the issue being scanned for. Some examples are be "Poor Error Handling" or "SQL Injection". This defaults to "General".
`$result->category_link`	This can be a URL that will be attached to the category specified above. There is no default URL, and failing to set one will affect the HTMLReport by having the category not be a link.
`$result->plugin_name`	This should be the name of the plugin that found the particular scan. It is displayed in an HTMLReport when the user hovers their mouse over the severity column. This defaults to the name of the class (`plugin_ssn_search`, in the case above).

`$result->source`	This is the line of source code or message that indicates the issue found. If the `line_number` is less than or equal to 0, then this value defaults to a blank (""). Otherwise, the value is taken from `file_contents` using `line_number`.
`$result->is_source_code`	If set to true, then the `$result->source` text is displayed in a fixed-width font. The default value is `false`.
`$result->source_context`	This specifies the context of the result. By default, this includes from three lines before to three lines after the `line_number` from `file_contents`.
`$result->custom`	This is an array that can be used to pass additional information back to the engine and subsequently to the report renderer. By default, this array is empty.

You can learn more about writing PHP plugins by examining the plugins included in the Yasca distribution. You can learn even more by examining the source code within the source distribution.

3.3.2 Writing a New Grep Plugin

Grep plugins are much easier to write than PHP plugins, but are not as powerful. In this section we will create a Grep plugin to perform the same function as the plugin_ssn_search class defined above.

Grep plugins are defined in files with the extension .grep and are managed by the Grep.php plugin. To avoid confusion, we will call the Grep.php plugin the "Grep plugin" and the .grep files "Grep scripts".

The following grep script is functionally equivalent to the plugin_ssn_search plugin that we developed in the last section. It scans all files that match the specified extensions for the same regular expression as before.

```
001 name = Social Security Number
002 file_type = java,properties
003 grep = /[^\d]\d{3}-?\d{2}-?\d{4}[^\d]/
004 category = Social Security Number
005 severity = 5
```

The grep expression is used internally by the Grep Plugin in a `preg_grep()` PCRE function. Internal option specifiers such as /i or /U can be included. More information about syntax is available in the PHP manual.

A number of grep scripts are included in the Yasca distribution. These are:

- Ajax: Detects use of AJAX in JSP, JavaScript, or HTML files
- Console.Output: Using System.out or System.err
- Crypto.MD5: Use of MD5 (deprecated hash algorithm)
- CustomCookies: Inserting or modifying client-side cookies.

- DoS.ReadLine: Using BufferedReader.readLine(), which blocks until EOF is read
- General.BadLanguage: Searches for bad language
- General.Password.Hardcoded: Looks for hardcoded passwords in source code
- Information-Disclosure.Comment: Comments in JSP or HTML files
- Injection.SQL: SQL Injection
- Injection.XSS.JSP: Cross-Site Scripting via <%=request.getParameter("foo")%>
- Licensing: GNU or other public licenses
- Process.exec: Executing external processes
- Process.ForName: Instantiating classes on the fly
- Process.LoadLibrary: Use of JNI
- Stability.Sleep: Sleeping within a servlet
- String.equals: Using == instead of .equals() to compare Strings.

3.3.3 Writing a New PMD Ruleset

PMD Rulesets are easy to write and extremely powerful.

```
001 <rule name="SocialSecurityNumber"
002       message="Don't use social security numbers."
003       class="net.sourceforge.pmd.rules.XPathRule"
004       externalInfoUrl="#">
005     <description>Do not use SSN in source code.</description>
006     <priority>2</priority>
007     <properties>
008         <property name="xpath">
009             <value>
010 <![CDATA[
011 //Literal[matches(@Image, "[^\d]\d{3}-\d{2}-\d{4}[^\d]")]
012 ]]>
013             </value>
014         </property>
015     </properties>
016     <example>
017         String s = "123-45-6789";
018     </example>
019 </rule>
```

The abstract syntax tree that PMD creates during compilation is actually XML, which can be queried using XPath. In the example above, the XPath expression is used to locate all literal values that match the regular expression for social security numbers.

3.3.4 References

PMD

- PMD Home Page: http://pmd.sourceforge.net/
- PMD Applied (book): http://www.pmdapplied.com/
- XPath Tutorial: http://pmd.sourceforge.net/xpathruletutorial.html
- XPath 2.0 Specification: http://www.w3.org/TR/xpath20/

Grep

- Regular Expression Functions (PCRE) in PHP: http://us.php.net/manual/en/ref.pcre.php
- PCRE Information: http://www.pcre.org/pcre.txt

4 Known Issues

4.1 Known Bugs

ID	Severity	Description
BUG-001	Medium	JLint does not return the correct path when analyzing Java source code.
~~BUG-002~~	~~Medium~~	~~PMD long descriptions and examples are not included.~~
BUG-003	Medium	PMD returns errors when using the basic-jsp ruleset and scanning Java source code.

4.2 Future Enhancements

ID	Severity	Description
ENH-001	Low	Adapt to work on non-Windows systems (executable translation table?)
~~ENH-002~~	~~Medium~~	~~Allow re-mounting of links to a local file system (so reports can be viewed on another workstation where the target files are some place else).~~
~~ENH 003~~	~~Medium~~	~~Allow modifications to descriptions and severities~~